

Real Time Coincidence Processing Algorithm for Geiger-Mode Ladar using FPGAs

Rufo A. Antonio¹, Alexandru N. Vasile¹, Muller Lon¹, Francis Thomas¹

¹Massachusetts Institute of Technology - Lincoln Laboratory, Lexington, MA, USA

Abstract. This paper introduces the first ever Geiger-mode ladar processing algorithm that is suitable for implementation on an FPGA enabling real time processing and data downlink. Current airborne Geiger-mode ladar systems obtain high resolution and wide area coverage, but require sensors and processing counterparts with large size, weight, and power (SWaP) requirements. We are developing an airborne “Micro-ladar” system that collects up to 1 billion samples/sec, weighs on the order of 250 grams, consumes no more than 20W and fits in a package not much larger than a coffee cup (320cc). To reduce SWaP, we developed embedded FPGA real time processing algorithms that take noisy raw data, streaming at upwards of 1GB/sec, and filters the data to obtain a nearly noise-free 3D point cloud with high compression rates, resulting in a 2MB/sec output data rate that can be readily downlinked to the ground over typical UAV communication links. A physical 64x256 Geiger-mode ladar array was integrated with an FPGA processing board running a baseline processing algorithm where the processing board has 8 orders of magnitude lower SWaP (m³kgW) than typical airborne ladar processing systems. Quantitative results using simulated ladar data input indicate that the new FPGA algorithm produces data with quality comparable with previous state of the art 3D ladar processing algorithms while suffering a reduction in area coverage rates.

1 Introduction

3D ladar imaging sensors are critical enablers for autonomous navigation systems in robotic and unmanned aerial vehicle (UAV) applications, as well as for airborne wide-area 3D terrain mapping and foliage penetration (FoPen) missions in support of national security (Albota M. H., 2002). In this paper, we develop the first ever Geiger-mode ladar processing algorithm implemented on a FPGA. By utilizing FPGAs we are able to achieve a “micro-ladar” system that resides in a design space not previously explored. It has sampling rates on the same order scale as the much larger airborne ladar system, ALIRT, but operates at shorter ranges measured in hundreds of meters instead of kilometers (Knowlton, 2011).

The rest of the paper is organized as follows. In section 2 we review related work that is essential in the implementation process. Section 3 discusses the different FPGA algorithm implementations. In Section 4 we present our implementation and results including FPGA complexity studies and algorithm performance results. Detailed FPGA utilization reports are generated for current operating parameters and extrapolated for future system upgrades. Simulated ladar data is processed using the FPGA

DISTRIBUTION STATEMENT A. Approved for public release: distribution unlimited.

This material is based upon work supported by the Assistant Secretary of Defense for Research and Engineering under Air Force Contract No. FA8721-05-C-0002 and/or FA8702-15-D-0001. Any opinions, findings, conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Assistant Secretary of Defense for Research and Engineering.

and compared to the Matlab output (Michael E. O'Brien, 2005). Section 5 concludes with a discussion of the lessons learned and directions for future work.

2 Related Work

Three-dimensional Laser Radar (3-D Lidar) sensors output range images, which provide explicit 3-D information about a scene (Heinrichs, 2001). Most 3D imaging for robotics rely on synchronous time of flight (TOF) focal plane arrays (FPAs), with one example of that being the Microsoft Kinect sensor. Such sensors typically work up to 10m in range, limited lighting conditions (indoor), and take on the order of 100k to 1million measurement samples per second. Pulsed TOF FPAs, such as Velodyne Lidar, are used for precision surveys and autonomous vehicle navigation, with ranges upwards of 1km and 100k to 1 million measurement samples per second. Next up are pulsed, flying spot single pixel linear mode sensors, with increased range to target, but still limited measurement rates fewer than 1 million samples per second. Such systems are used for mapping applications, however their area coverage rates border on the low side for collecting high resolution imagery over large regions. Figure 1 provides some background on where prior 3D imaging systems map in terms of measurement sampling rates versus range to target.

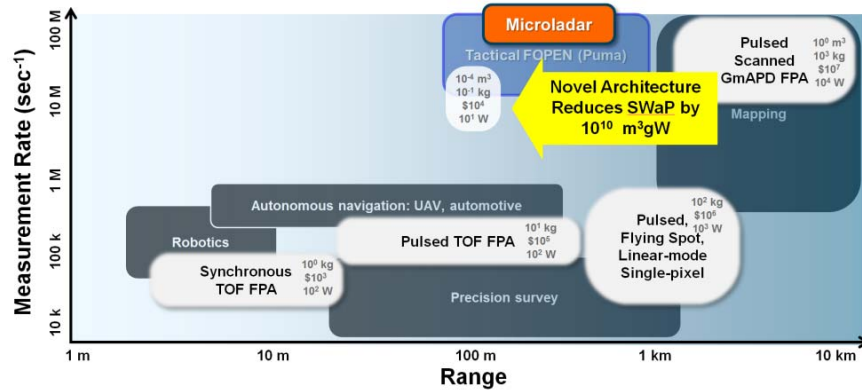


Figure 1. Background on where prior 3D imaging systems map in terms of measurement sampling rates versus range to target.

Pulsed Geiger-mode APDs, especially combined with scanning hardware, offer a significant step-up in both measurement rates (100 mil samples per second) and range to target (10km+), thanks to ever larger multi-pixel arrays as well single-photon sensitive detection (Albota M. A., 2002) (Aull, 2002). An example of such a system is ALIRT (~10-20 million samples/sec, 7km range) (Knowlton, 2011).

The proposed “micro-ladar” system resides in a location not previously explored in this parameter space. It has sampling rates on the same order scale as ALIRT, but at shorter ranges to target, hundreds of meters instead of kilometers (Marino, 2003). To be able achieve this measurement rate while requiring a many-order reduction in size-

weight and power (SWaP) necessitates a radical rethink of the algorithms used to process the data.

3 FPGA Implementation of Noise Filtering Algorithm

In this section we introduce and implement novel Geiger-mode lidar de-noising algorithms (PixelCP) developed for Field Programmable Gate Arrays (FPGAs).

3.1 Algorithm Design

A new set of 3D lidar noise filtering algorithms must be developed in order to achieve such a drastic reduction in SWaP while maintaining performance. The algorithms were first implemented in Matlab where multiple modules of varying complexity were developed, leading to a large number of possible module combinations (384 variants). Figure 2 captures a high-level description of each processing step as well as implemented modules. This paper explores the implementation of a modified “baseline” algorithm.

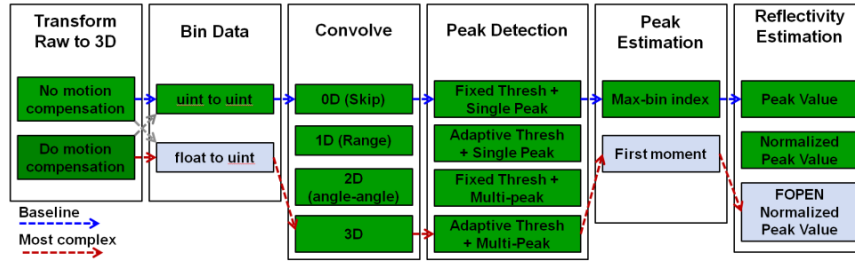


Figure 2. PixelCP Variants.

For the first step, transforming the data, we have two options: motion compensation enabled or disabled. For the no-motion compensation option, we are assuming the sensor movement is negligible and that each APD pixel captures the same scene location. The data is transformed from raw range data directly to a 3D angle-angle-range (AAR) space that is defined in angle-angle by the APD row-col information. In essence, we are taking the blur hit due to any jitter or unexpected movement during that integration time. For the motion compensation case, we developed a method that is computationally much less complex than the transform codebase (Vasile, 2012) used on prior 3D lidar sensors. The reason for this is that such transforms require a lot of double precision floating point math, which are not well suited to FPGA implementation. Thus, we developed an approximate transform method, where we define a 3D angle-angle-range space in the APD row-col-range space of the first frame for a given integration period. The rest of the APD data frames within an integration interval are motion compensated compared to first-frame AAR space, by first computing the relative rotation-translation compared to the first frame, and then applying that projection matrix to just the center line of sight (LOS) at average range for that cur-

rent frame, in order to obtain an delta row-col-range translation. This motion compensation transformation from raw stream to a 3D AAR space is exact for the center line of sight, but only approximate for the remaining pixels under certain conditions. Given no sensor translation and just roll-pitch rotational changes (due to mirror scanning, platform attitude), the 3D transformation is equivalent to a full per-pixel transform. With sensor translation and/or yaw rotation (rotation around the center LOS), the motion compensation transform is only an approximation of the full per-pixel transform. For translation-only motion, these approximations are present due to variation in range position and are negligible considering that the range to target (300m) and range gate (~100m) are much bigger than the platform movement (15m/sec) over an integration time upwards of 100ms, which leads to 1.5m of movement. In essence, 3D Cartesian translation errors can be well approximated as 3D AAR space translations through the use of small angle approximations. However, the motion compensation approximation is very sensitive to yaw-rotations (around the center LOS) with the array size amplifying the effect: even for small sub-pixel pitch angles ($<100\mu\text{R}$), the lever arm effect for an APD corner pixel compared to the center pixel can lead to significant blur. For example, a $50\mu\text{R}$ (1/2 pixel error) yaw rotation would lead to a blur on the corner of a 256×256 APD array of $50\mu\text{R} * (1282+1282).5 / 100\mu\text{R} = 90$ pixel error. Considering that most attitude changes for a Puma-like fixed wing UAV occur in roll-pitch rather than yaw, and mirror scanning typically can be expressed as purely roll-pitch changes, the sensitivity to yaw changes is not concerning. If we decide to use a different platform, such as a quad-copter, the motion-compensation might need to be improved. For now, we will implement this approximate motion compensation technique, as the computation is simple and the approximation holds well for our target platform.

Now that we have data converted to a 3D AAR space, we attempt to automatically distinguish signal from noise using the spatial coincidence of points. The concept is fairly simple to grasp: the more 3D points returned at the same spatial location, the more it is likely that the points came from a real scene surface as opposed to “hits” due to background light or dark noise. To detect spatial coincidence, we bin data into 3D voxels of various volumes in terms of pixel-pixel-range units. Smaller bin sizes allow for higher frequency data to be captured, at the expense of increased ability to detect signal from noise and lower memory storage requirements. Larger bins recover signal better in present of noise and lead to lower memory storage, but lead to reduction in 3D data fidelity. The best of both worlds (high fidelity and good SNR) can be achieved by convolution.

For small bin sizes, where we have high 3D fidelity, the SNR is typically not enough to discriminate signal from noise. Convolution is used to recover spatial coincidence, while still preserving 3D data fidelity. For small bin sizes, 3D convolution is applied, where a 3D matched kernel is convolved with the 3D binned array. This matched kernel typically takes on the shape of the system point spread function (PSF), which captures sensor uncertainties in the angle-angle and range directions. The kernel can be a full 3D kernel, but might also consist of just a 2D angle-angle kernel, or potentially just a 1D range kernel, depending on the angular error to angular bin size ratio and range error to range bin size ratio.

Once coincident information is available, we can now detect valid signal as 3D peaks. The peak estimation algorithm has four algorithm variants to choose from.

The first is single peak option and fixed threshold. In this method, a predetermined noise threshold is set for the minimum bin value. The values for each unique angle-angle bin are considered, to extract a 1D histogram in the range direction. For each histogram, the index that contains the maximum bin value is recorded as the single-peak range position, with the bin value directly determining the 3D point reflectivity. The result is a 2 ½ D single peak range image. The adaptive threshold and single peak method is similar to the first approach, but now adaptively determines a threshold based on overall signal integration, bin size as well current data strength. The third algorithm, a fixed threshold multi-peak method, outputs both the first peak as well as the last peak above a certain threshold, leading to a 2-peak solution. The most advanced algorithm uses a non-maxima suppression technique to identify peaks maxima and ignore peak-begin and -end tails, and outputs all such peak locations above an adaptive threshold.

The peak estimation algorithm ties closely with the peak detection algorithm. The first method considers the index for the maximum bin value as the resulting output range position. This leads to gridding and discretization in range results, with flat surfaces appearing jagged. The second method takes into account neighboring angle-angle-range bins to obtain a floating point estimate in both angle-angle and range, removing some of the gridding artifacts.

Lastly the reflectivity algorithm has three methods to choose from. The first method assigns the reflectivity based directly on the bin value. The second method takes into account integration time to obtain what is better known as “probability of detection”. The third method is based off of the second method, but takes into account the effects of APD and photon blocking effects for surfaces under foliated conditions.

In order to evaluate algorithm performance and capture image quality as a function of different data collection parameters, we need quantitative image quality metrics. For this analysis we observe angular resolution, range precision, impulse detection, edge sharpness, FOPEN ground coverage, and SNR.

3.2 Single Peak Detection

We first implemented a very basic PixelCP algorithm directly in VHDL. For this we selected module variants that would be easily parallelized and implemented on a FPGA. We chose not to incorporate motion compensation in our implementation because its complexity and benefits were deterministic of the devices used for measuring motion. For binning data we used a uint to uint operations while using eight by eight “macro-pixels”. This meant that each group of 64 pixels would produce one output image pixel. In the next step we chose to forgo convolution as it is the most complex operation. For the peak detection, we used fixed-threshold, single-peak, and max-bin index methods. Lastly we implemented the normalized peak value algorithm for the reflectivity estimation.

The first step was to think of a way to highly parallelize this combination of PixelCP modules. We found that each “macro-pixel’s” operations were done completely independently of each other and therefore would be the best place to start parallelizing the algorithm. Next, the range value from each of the 64 pixels inside the macro-pixel is used as the address for the bin memory. The value of each pixel is read sequentially

as the address for the bin memory. The value at that address is then incremented by one and written back into the bin memory at that same address. Additionally, when the value in the bin memory is incremented by one it is also compared to the last known highest value. If the new value is equal to or greater than the previous value, it is stored in an output memory block, “Max Hits”. We then check to see if the value has been changed and if so the bin address that corresponds to that value is also stored in another output memory block, “Max Range”. A block diagram of the operations performed on one macro-pixel can be seen below in **Error! Reference source not found.**

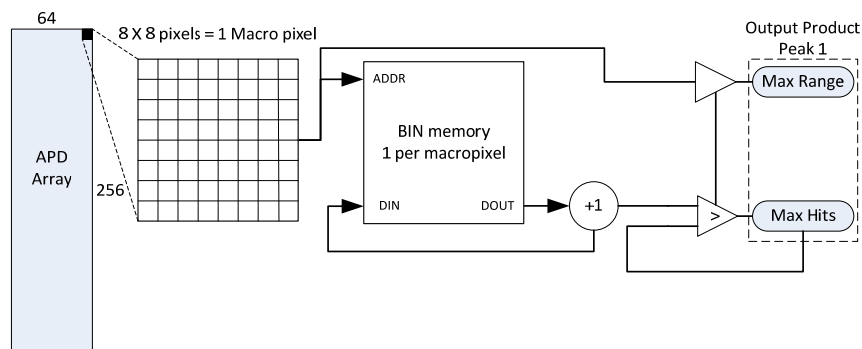


Figure 3. Single peak block diagram.

Over the entire array 256 individual macro-pixels are operated on in parallel and therefore produce one output as quickly as possible. In addition to parallelizing the macro-pixels we also implemented the firmware to operate as data was being read in from the array. For example if the array was reading data from four quadrants at the same time each quadrant would be operated on as the data flowed in real time. This allows us to pipeline the processing with the data input so that there is no delay from data gathered to processed output.

In order to simplify timing and synchronization between the FPGA and the APD the FPGA clock is operated at 125 MHz, the same rate as the APD data readout. We can then use our knowledge of FPGA components to calculate an expected completion time. Reading all 64 pixels into memory takes 2 clocks per pixel, for a total of 128 clocks. Incrementing the value and storing it back in memory takes another 2 clocks. All together this makes 256 clocks per output. If all the macro-pixels are operated on in parallel then the output should be ready in 2.048 microseconds. This is far faster than the 20 kilohertz (50 microseconds) frame rate of the system.

3.3 Multi-Peak Detection

Next, we implemented a second/multi-peak implementation in order to achieve foliage penetration. We came up with a method that identified any number of additional peaks without the need to reprocess the input data. The first peak would be calculated exactly identically to the single peak version and then the bin memory would be reprocessed to produce a second peak.

The value of the max range output of the first peak is used as a starting address in the bin memory. From that point a predetermined range offset is subtracted from the address as to avoid any nearby false peaks. The value at this new memory address (range) is compared with the last known highest value. If the new value is equal to or greater than the previous value, it is stored in a second output memory block, “Max Hits”. If the value has been changed the bin address that corresponds to that value is also stored in another output memory block, “Max Range”. A block diagram of the operations performed on one macro-pixel can be seen below in Figure 4.

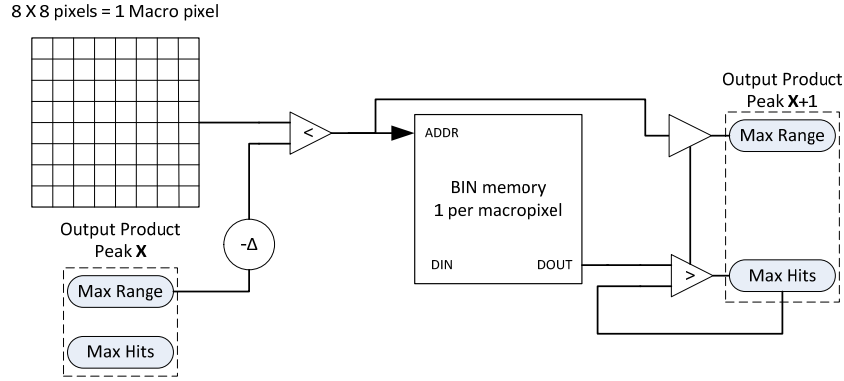


Figure 4. Multi-peak block diagram.

This method preserves the parallelized macro-pixels that were created in the first peak. We can then repeat this algorithm to allow for any number of additional peaks to be detected. However, these additional peaks require that the previous peak must be fully processed and therefore sequentially add time to the total process.

In order to simplify timing and synchronization between the FPGA and the APD the FPGA clock is operated at 125 MHz, the same rate as the APD data readout. We can then use our knowledge of FPGA components to calculate an expected completion time. For the second peak not all 64 pixels must be read from memory. The number can fluctuate depending on the location of the first peak. If we assume that the first peak is in the upper half of the range extent then we can also assume approximately 50 percent of the pixels will be read out. In this case there are only a total of 128 clocks per output. If all the macro-pixels are operated on in parallel then the output should be ready in 1.024 microseconds. This is far faster than the 20 kilohertz (50 microseconds) frame rate of the system.

4 Results

The following sections report on FPGA utilization, algorithm timing, and compare the outputs with known good outputs from Matlab. Reports were generated using Xilinx and Questa Modelsim development tools. All results were conducted on Kintex 7s with a clock speed of 125 MHz 64 by 256 array operating frame rate of 20.

4.1 Timing Analysis

We closely analyzed Modelsim simulations to determine the performance of both the single and multi-peak algorithms. For the single peak algorithm we found that the processing could be conducted as the raw data was read in. This removed the wait for incoming data to finish before processing. Additionally, we reduced the number of macro-pixels operating in parallel as most were idle while waiting for data to come in. This method reduced FPGA complexity while maintaining timing; however it did serialized some of the processing steps. Below is a timing diagram that shows the algorithm implementation on a FPGA. The timing markers in this figure are used as approximate time identifiers and not for exact timing values.

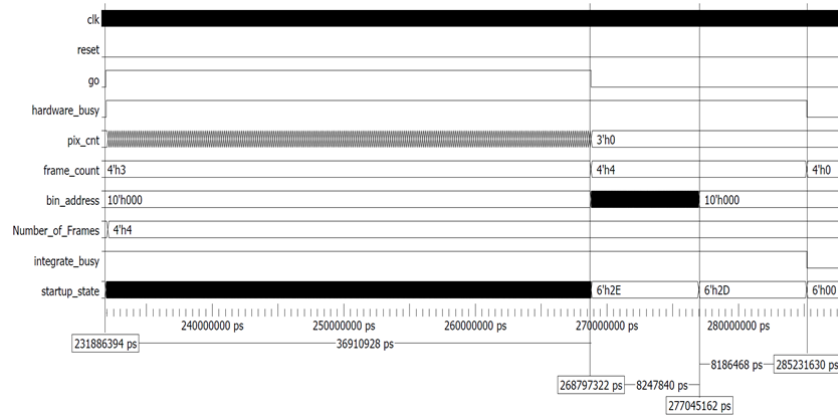


Figure 5. Multi-peak Timing.

When looking at Figure 5, the signal “frame_count” identifies the current frame that is being processed and “pix_cnt” identifies the computation of the first peak. The timing marker on the far left (231886394 picoseconds) identifies the start of data being read into the FPGA. The second marker (268797322 picoseconds) identifies the 37 microseconds that it takes to read the data in and process the first peak output. While the timing is significantly longer then what we had calculated in section 3.2, it still exceeds system requirements.

For the multi-peak implementation we chose to build off of the single peak implementation. Keeping the partially serialized macro-pixels did slow the second peak output, but still falls well within the timing requirements of the system. In Figure 5 the second peak processing is identified by the signal “bin_address”. The 8.2 microseconds between the second marker (268797322 picoseconds) and the third marker (277045162 picoseconds) shows the generation of the second peak output directly after the first peak processing finishes. After both the first and second peak are both calculated there is still another 8 microseconds of idle time before the next frame is ready to process. Although we didn’t implement the functionality there is almost enough time to perform a third peak calculation and still meet system requirements.

4.2 Algorithm Performance

We defined system requirements in terms of 3D spatial fidelity of processing results. Prior studies of human interpretability of foliated scenes performed for FALCON-I as well as ALIRT indicate an x-y post spacing of 25cm or higher fidelity is needed to accurately identify man-made structures versus vegetation clutter. Thus, we require that 3D snapshot products retain features on the order of 25cm x-y spatial resolution. Current technology enables a range resolution around 25cm, which sets the desired range resolution requirement. Thus, we require a spatial resolution of 25cm in all three dimensions. MAPCP and MPSCP

Table 1. PixelCP vs. MPSCP Performance

	Angular	Range	Edge	FOPEN Ground	SNR	Data Redux Clear FOPEN	Data Compres- sion
PixelCP	1267uR	0.19m	0.3m	31%	$0.9 \cdot 10^5$	187x 101x	18x
MPSCP	600uR	0.09m	0.14	45%	$5 \cdot 10^5$	~100x 25x	~10x

Next, we tested the single and multi-peak design on a simulated data set. A small snapshot of the simulated raw input data can be seen below in Figure 6.

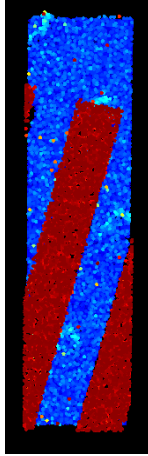


Figure 6. Raw Simulation Data.

The data set ran through the original Matlab code as well as a Modelsim simulation and the output images were compared using Matlab's point cloud tool. The outputs of both can be seen below in Figure 7.

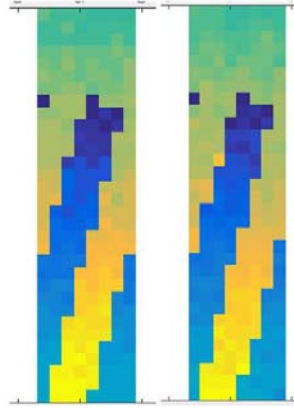


Figure 7. Matlab First Peak Value (Left) vs. FPGA Single/Multi Peak Value (Right).

Upon inspecting the data sets we found small differences in the output images. However, with further investigation the two algorithms addressed the ambiguity of similar hit values in different range bins. In this situation the Matlab algorithm would select the bin with range 50m, however the FPGA algorithm would select the 20m range bin. This difference in algorithms is neglected as it still returns extremely similar results.

With verification that both the single and multi-peak implementations met timing and correctly generated output products on simulate data the next step was to test the algorithm on a live data stream. For this experiment the system was set up with a 64 by 256 GmAPD operating at a 20 kHz frame rate. Using a PicoQuant laser source a single pin hole was illuminated so that approximately 10 percent of the photodiodes were firing at any given time. This data was read out and processed by the FPGA in real time. The raw input and output products of both the first and second peak can be seen below in Figure 8.

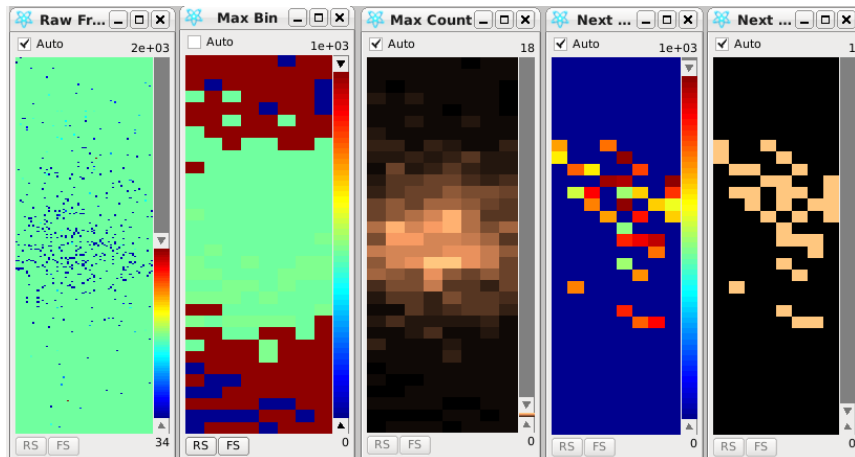


Figure 8. Live FPGA performance from left to right: raw data input, first peak range, first peak hit count, second peak range, and second peak hit count.

The image labeled “Max Bin” is the range output of the first peak; it clearly shows a significant difference in range directly around the center of the image (near the pin hole). The next image to the right displays the hit counts associated with that range; a clear image of the laser profile can be seen. The fourth and fifth images show the max range and hit count of the second peak. However, since only one laser pulse is present during each frame the data in these outputs is an artificial product of random noise on the array.

4.3 FPGA Utilization

Furthermore, we generated FPGA utilization reports using Vivado to determine what FPGA could accommodate these algorithm’s demands. We elected to skip the single peak only version due to the algorithm and timing performance of the multi-peak version. Vivado’s report gives us a detailed breakdown of FPGA usage in terms of bits used in each of the main FPGA blocks (Flip Flops, LUT, LUTRAM, BRAM, IO, GT, BUFG, CMT, and PCIe interfaces). As shown in Figure 9 the report was generated for a 256 by 64 size array and compared with a Kintex 7 FPGA.

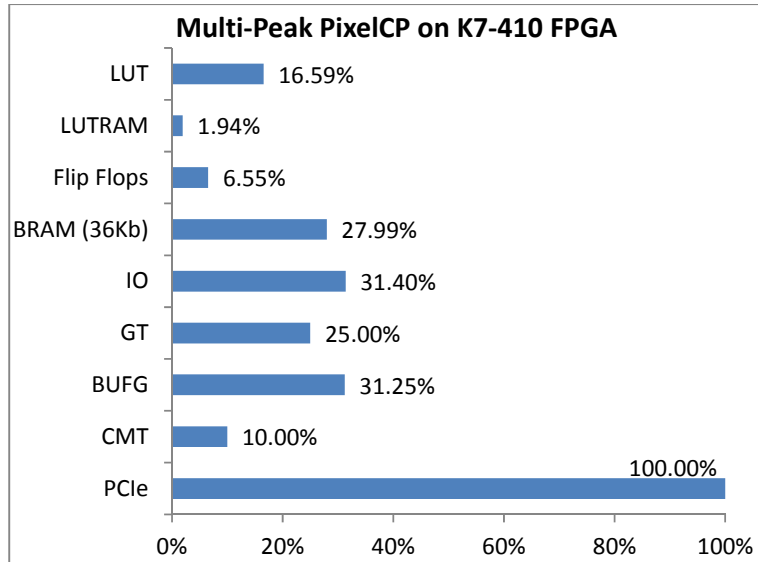


Figure 9. Multi-Peak PixelCP on K7-410.

From this report we can see that the multi-peak implementation is utilizes significantly less of the FPGA in every category compared to the convolution method. The PCIe utilization is maxed out at 100 percent because the specific FPGA we selected

only has one port available. For the system we designed this number would not expand with more complicated algorithms or larger arrays and therefore can be ignored.

It is apparent from this report that moving to a larger array format, such as a 256 by 256 array would still be plausible on this FPGA. If all resources were assumed to double with the array size the FPGA would still have resources to spare. However, further investigation would need to be done for the timing performance.

The SWaP of the processing component of the overall system can now be confidently reduced while still maintaining system performance. As seen in Table 2, there is a $3 \times 10^7 \times$ reduction from the current state of the art (MPSCP) processing SWaP to the new “micro-ladar” implementation.

Table 2. PixelCP vs. MPSCP SWaP

Algorithm	Size	Weight	Power
PixelCP	20 in ³	<1 lbs	~6 W
MPSCP	4107 in ³	270lbs	4000W

5 Conclusion and Future Work

In this paper we developed a coincidence processing algorithm to create high resolution, FOPEN imagery on low SWaP Geiger-mode lidar systems using FPGAs. We implemented a multi-peak and convolution version of PixelCP on a FPGA. We examined the performance of these algorithms when compared to their Matlab counterparts. Additionally determined their ability to operate with real time performance and presented the feasibility of implementing these algorithms on existing FPGA technology. Finally, we developed an algorithm that met all our system requirements.

Future work involves increasing the clock speed of the FPGA in order to reduce multi-peak timing. Additionally, a larger array can be implemented to cover larger areas or increase resolution. Lastly, a VHDL implementation of the convolution algorithm with simulated timing needs to be developed to further identify its validity on an FPGA.

References

- Albota, M. A. (2002). Three-Dimensional Imaging Laser Radars with Geiger-Mode Avalanche Photodiode. *MIT Lincoln Laboratory Journal* 13(2), 351-370.
- Albota, M. H. (2002). Three-Dimensional Imaging Laser Radar with a Photon-Counting Avalanche Photodiode Array and Microchip Laser. *Applied Optics* 41(36), 7671-7678.
- Aull, B. L. (2002). Geiger-Mode Avalanche Photodiodes for Three-Dimensional Imaging. *MIT Lincoln Laboratory Journal* 13(2), 335-350.
- Fouche, D. (2003). Detection and False-Alarm Probabilities for Laser Radars That Use Geiger-Mode Detectors. *Applied Optics* 42(27), 5388-5398.

- Heinrichs, R. A. (2001). Three-Dimensional Laser Radar with APD. *SPIE*, vol. 4377, 106-117.
- Keicher, A. B. (2000). Development of Coherent Laser at Lincoln Laboratory. *MIT Lincoln Laboratory Journal* 12(2), 383-396.
- Knowlton, R. (2011). *Tech Notes*. Retrieved 2016, from MIT Lincoln Laboratories: http://www.ll.mit.edu/publications/technotes/TechNote_ALIRT.pdf
- Marino, R. S. (2003). A Compact 3D Imaging Laser Radar System Using Geiger-Mode APD Arrays: System and Measurements. *SPIE* vol. 5086, 1-15.
- McIntosh, K. D. (2002). InGaAsP/InP Avalanche Photodiodes for Photon Counting at 1.06 μm . *Applied Physics Letters* 81, 2505-2507.
- Michael E. O'Brien, D. G. (2005). Simulation of 3D Laser. *MIT Lincoln Laboratory Journal* 15(1), 37-60.
- Vasile, A. N. (2012). Advanced Coincidence Processing of 3D Laser Radar Data. *Advances in Visual Computing Lecture Notes in Computer Science*, 382-393.
- Zayhowski, J. (1990). Microchip Lasers. *MIT Lincoln Laboratory Journal* 3(3), 427-446.